Documentation for Laba#5

Zakharchenko Anna

In this document I will describe the functionality of my code and what each function does. At first, I will show the input and output of functions in my code.

At first the user begins by entering the values they desire and to conclude the input, they enter 0.

0)**CreateList(pnode& head, pnode& tail):**

This function is used to create a doubly linked list.

It takes two parameters, head and tail, which are passed by reference.

It initializes head and tail to nullptr.

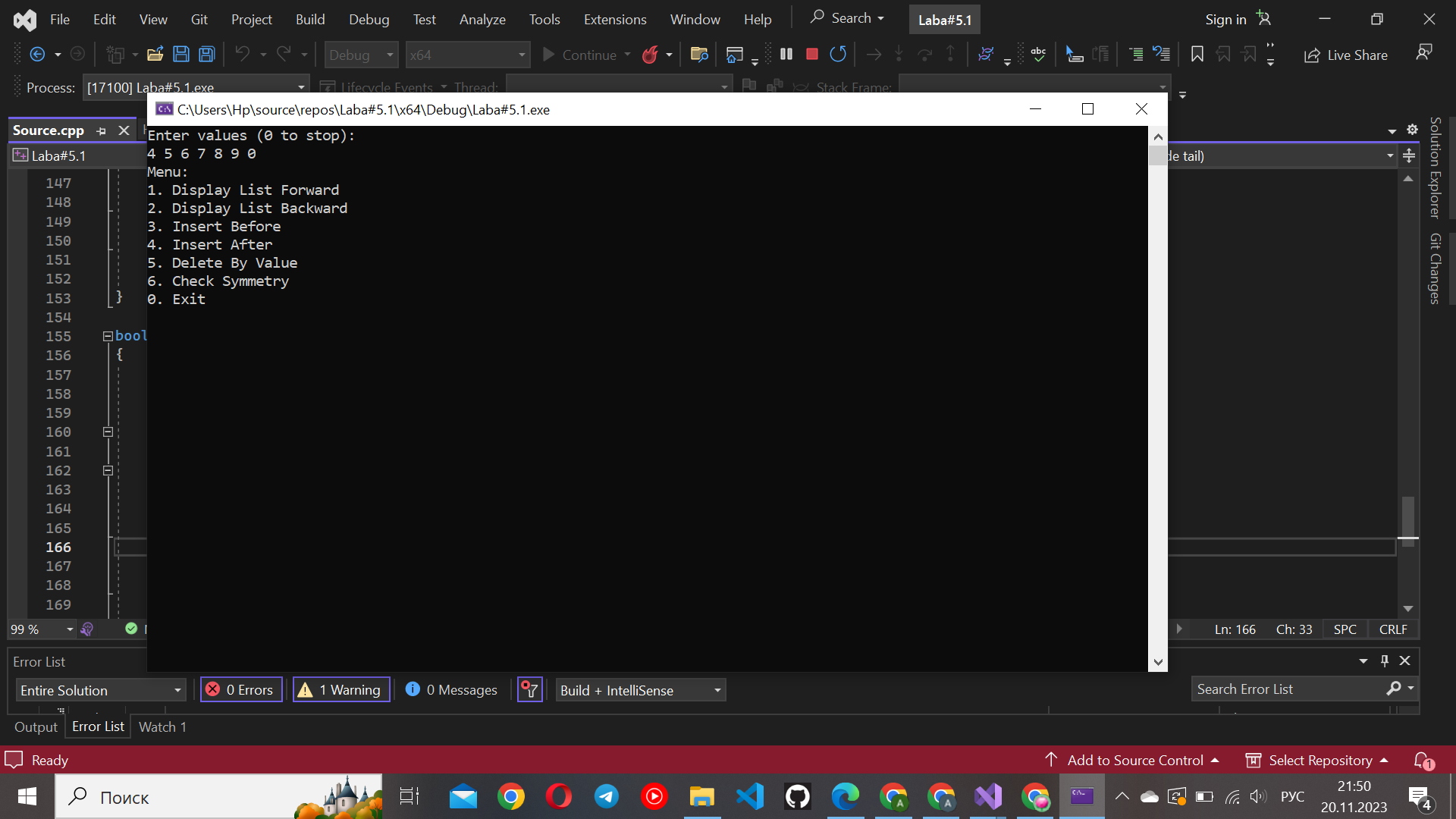
It then prompts the user to enter values for the nodes of the linked list until the user enters 0.

It dynamically allocates memory for each node, assigns the user input to the info field, and links the nodes accordingly.

Then it shows Menu in which we can see 6 options. I use a while loop for creating a menu.

The loop continues until the user chooses to exit (option 6). Depending on the users choice , it calls the function to persorm the desired operation on the linked list.

Then we have 6 options. The results of the operations can be viewed below.

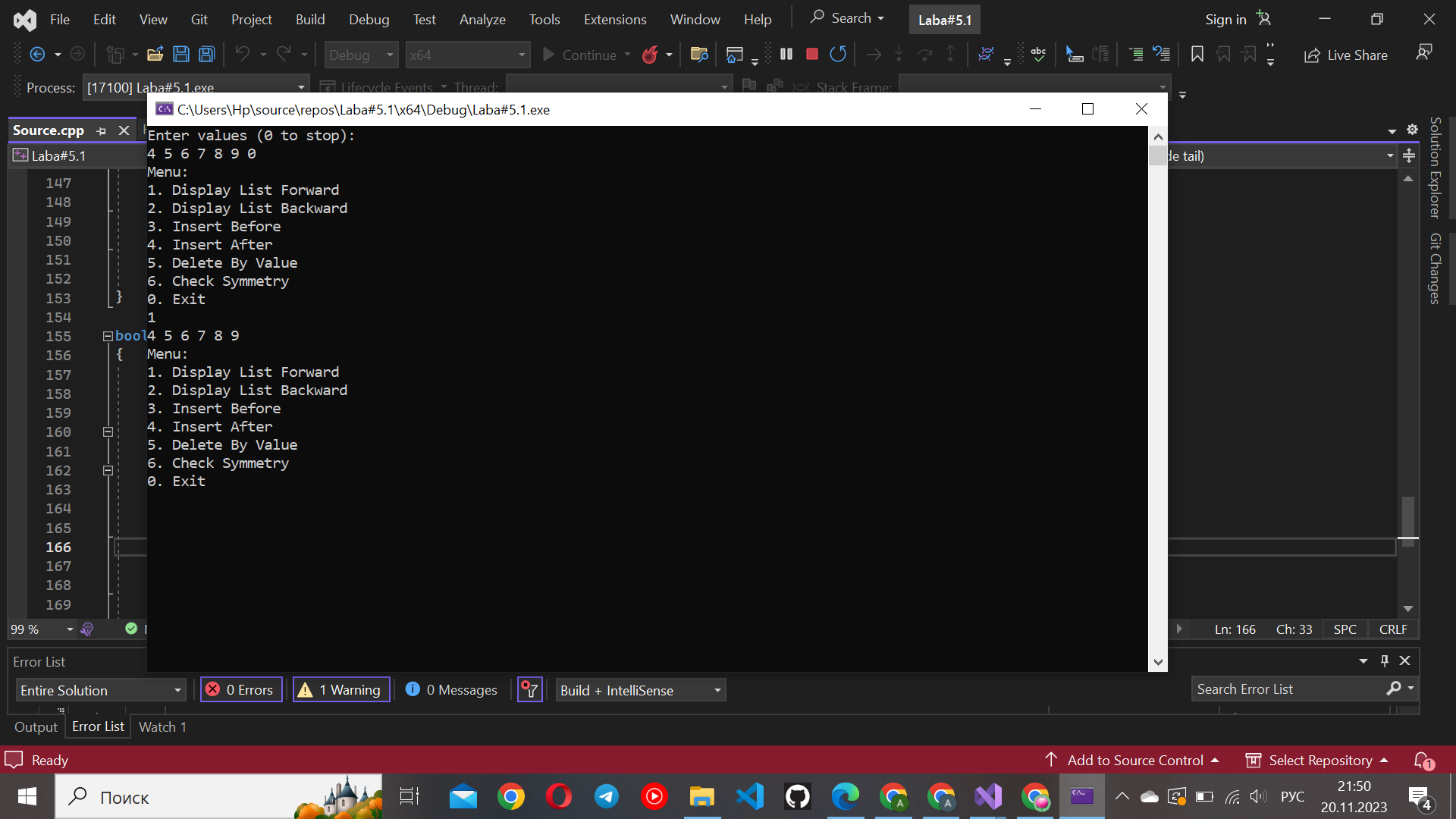


1)**DisplayList(pnode head):**

This function is responsible for displaying the elements of the linked list in the forward direction.

It takes the head of the list as a parameter.

It traverses the list from head to the end (nullptr) and prints the info field of each node.

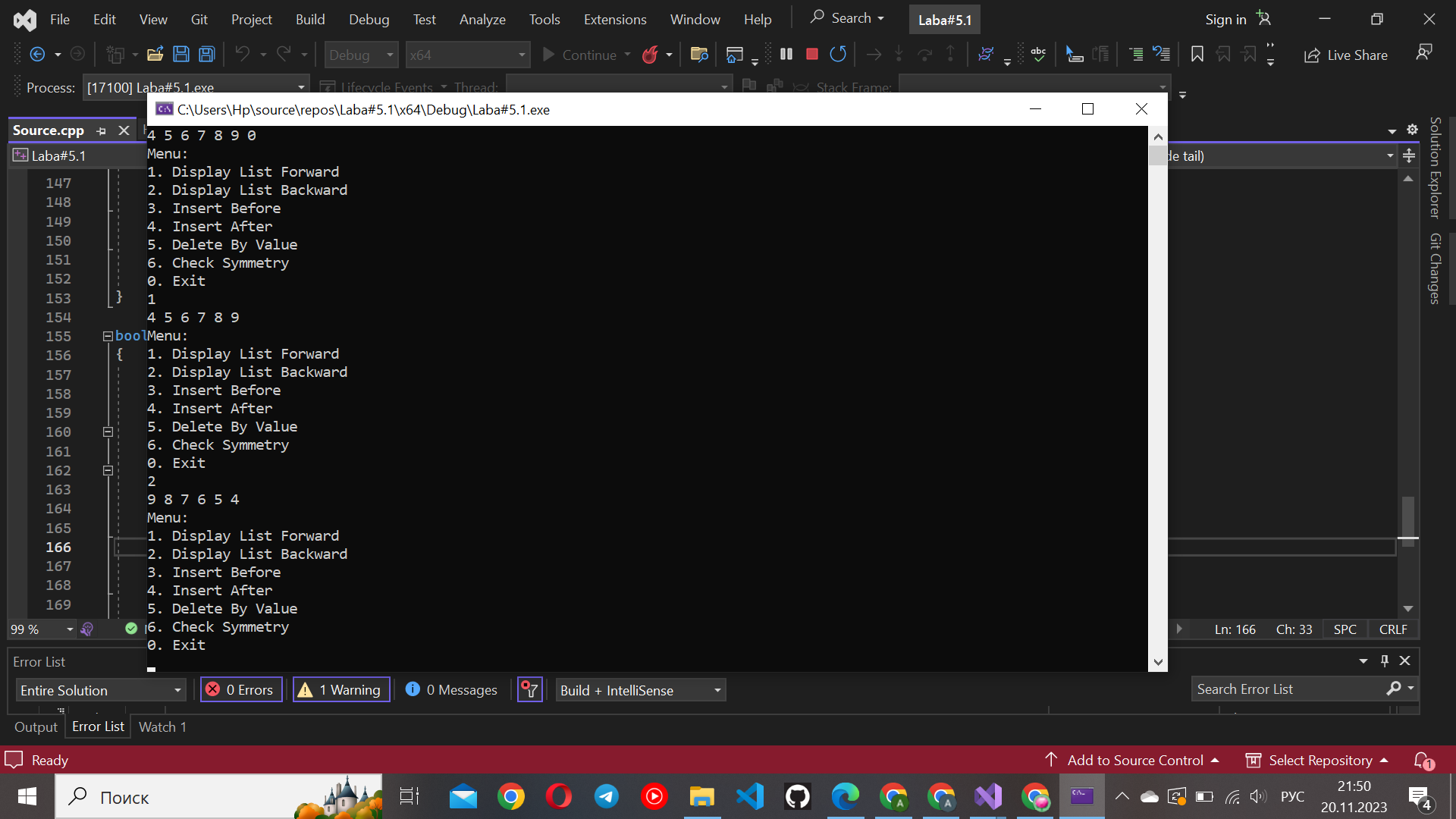


2) **DisplayReverseList(pnode tail):**

This function is similar to DisplayList but displays the elements of the linked list in reverse order.

It takes the tail of the list as a parameter.

It traverses the list from tail to the beginning (nullptr) and prints the info field of each node.



3) **InsertBefore(pnode& head, int info, int new\_info):**

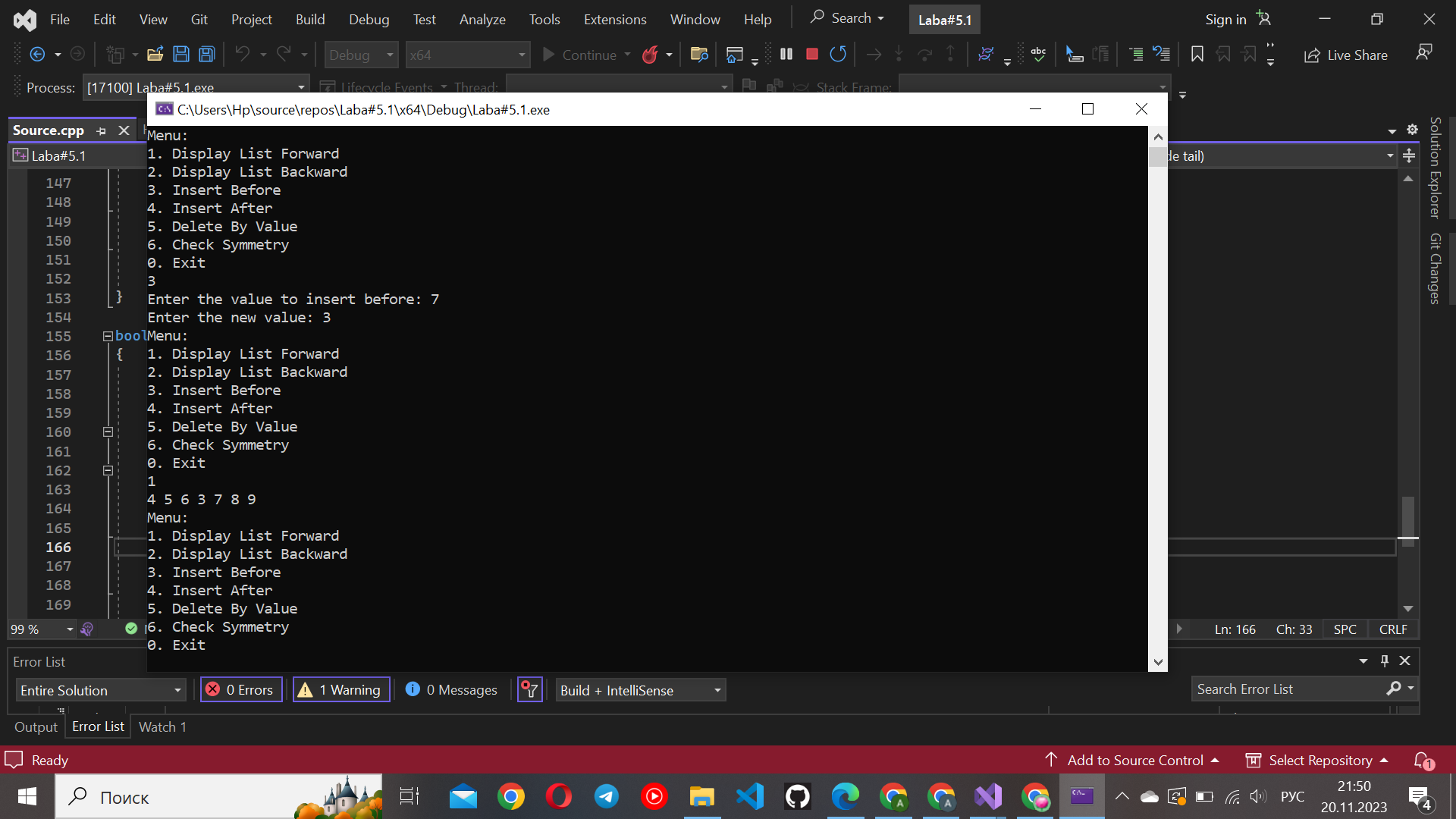
This function inserts a new node with the value new\_info before the node containing the value info.

It takes the head of the list, the value to find (info), and the value to insert (new\_info).

It checks if the list is empty. If it is, it returns 0.

It then traverses the list, finds the node with the value info, and inserts a new node with the value new\_info before it.

Here, the user can choose which value he/she want to insert and before which value.



4**) InsertAfter(pnode& head, pnode& tail, int info, int new\_info):**

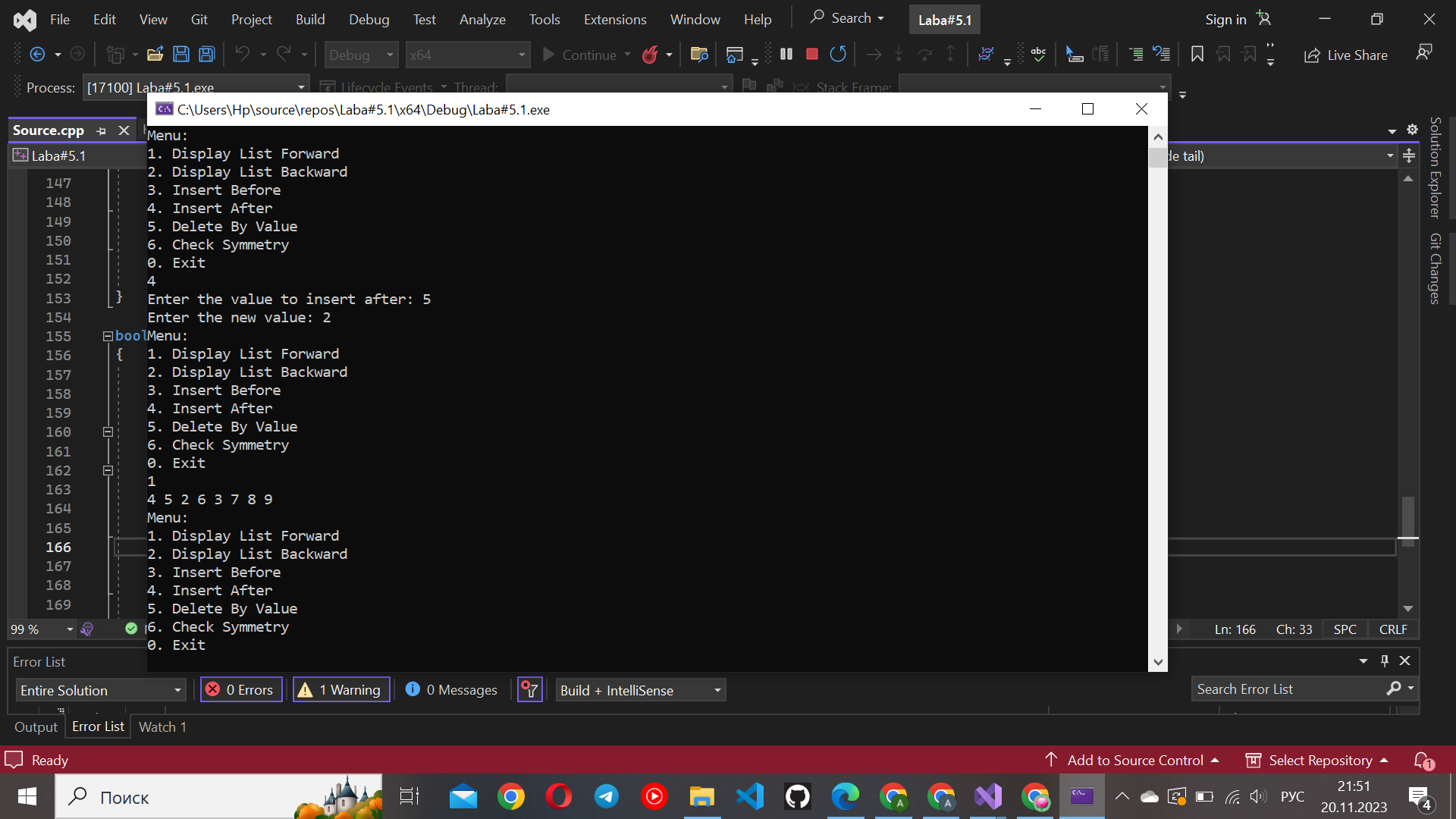
This function inserts a new node with the value new\_info after the node containing the value info.

It takes the head and tail of the list, the value to find (info), and the value to insert (new\_info).

It checks if the list is empty. If it is, it returns 0.

It then traverses the list, finds the node with the value info, and inserts a new node with the value new\_info after it.

Here, the user can choose which value he/she want to insert and after which value.



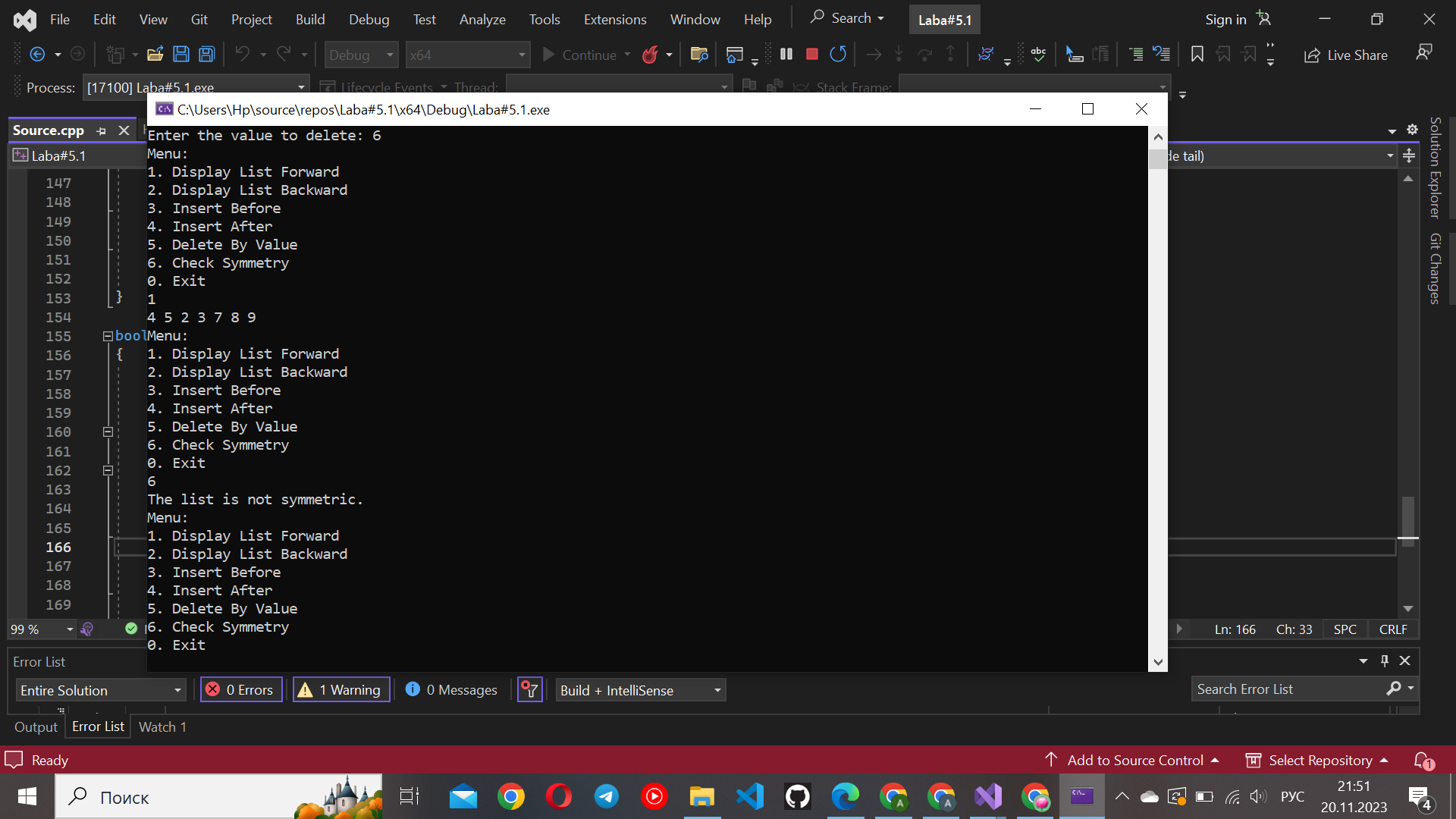
5) **DeleteByValue(pnode& head, pnode& tail, int info):**

This function deletes the first occurrence of a node with the given value (info).

It takes the head and tail of the list and the value to delete (info).

It checks if the list is empty. If it is, it returns 0.

It then traverses the list, finds the node with the value info, adjusts the links, and deletes the node.



6) **CheckSymmetry(pnode head, pnode tail):**

This function checks whether a doubly linked list is symmetric or not.

It takes the head and tail of the list as parameters.

It uses two pointers, forward starting from head and backward starting from tail, moving towards each other.

It compares the values of corresponding nodes in the forward and backward directions. If any pair of values differs, it returns false. If the entire list is symmetric, it returns true.

